富士通のスーパーコンピュータ : 科学技術計算用計算機高速化アーキテクチャの進歩(スーパーコンピュータのための数値計算アルゴリズムの研究)

AUTHOR(S):

内田, 啓一郎

CITATION:

内田, 啓一郎. 富士通のスーパーコンピュータ : 科学技術計算用計算機高速化アーキテクチャの進歩(スーパーコンピュータのための数値計算アルゴリズムの研究). 数理解析研究所講究録 1987, 613: 1-11

ISSUE DATE:

1987-03

URL:

http://hdl.handle.net/2433/99803

数理解析研究所講究録
613巻 1987年 1-11

富士通のスーパーコンピュータ
——科学技術計算用計算機高速化アーキテクチャの進歩——

富士通　電算機第一技術部

内田啓一郎

(Keiichiro Uchida)

科学技術計算用コンピュータの高速化アーキテクチャの進歩はFORTRAN DO LOOP処理の高速化の歴史である。逐次処理アーキテクチャにおいての並列化努力とその帰結としての並列処理アーキテクチャは，ハードウェア上のいかなる機能の改良からもたらされたかを分析する。それらのアーキテクチャ上の工夫が，富士通のスーパーコンピュータにどう適用されているかについて述べ，最後に今後の方向を示す。

1. 逐次処理アーキテクチャ

科学技術用計算機の高速化はFORTRANにおけるDO LOOPをいかに速く実行するかに注目して行われてきた。

1.1 レジスタの設置

初期の計算機はメモリとアキュムレータを持ったごく単純なものであったが，実数計算機用の浮動小数点演算器がハー

ドウェア化された。これに伴いメモリアクセスタイムが軽視できないようになり，ＣＰＵ内部に中間結果保持用としてレジスタが置かれ，メモリアクセスのボトルネックが軽減された。

またＤＯ　ＬＯＯＰでは，添字の加算が多重に行われる。これらをＣＰＵ内部レジスタとして保持するためインデックスレジスタを設置し，合わせてメモリオペランドのアドレス発生時にインデックスレジスタの値を加算した結果を送り，インデックス修飾のハードウェア化が行われた。これはアドレス計算用のハードウェアとしてはかなりの負担ではあるがＤＯ　ＬＯＯＰ性能向上に大きな効果があった。以上はプログラムに陽に見えるアーキテクチャ上の工夫である。

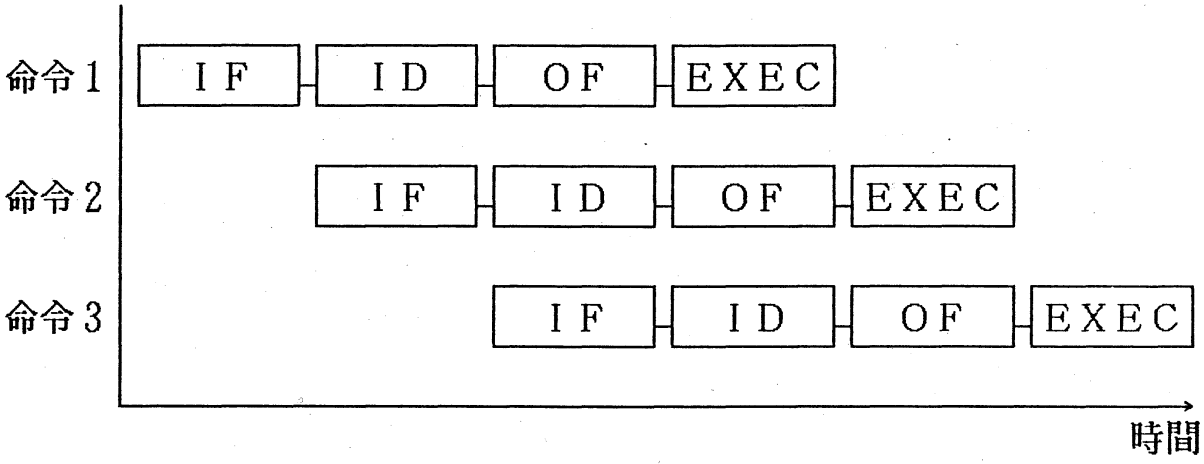
1.2　並列化動作と高速化設計技術

以下にはプログラムから陽に見えないハードウェア構成上の高速化技術について述べる。逐次処理アーキテクチャのままでの高速化は，各種動作の並列化であり，またデータの流れをスムーズにするための各種バッファの設置に帰着する。

並列化の第一は命令のパイプライン的制御である。命令実行を分業化し，個々のステージを専業化することにより，各ステージの実行時間で命令を実行して実効的な命令処理時間

を高速化する。究極的には1マシンサイクルに1命令の実行を行う機械があり，F230－75，M－780などがこの制御を採用している。

命令のパイプライン制御



さらにこのパイプライン制御を浮動小数点演算器にまで適用した機械があり，これらの機械では演算回路も専業化したステージを持っており，1クロックごとに演算結果を出力することができる。ここまで来ると演算オペランドを演算器に供給することもパイプライン制御を行う必要がある。つまりアドレス計算（インデックス修飾）を1クロックに1つずつ発生させ，メモリオペランドをパイプライン的に供給する必要がある。さらにメモリアクセスパイプラインも演算と同時に実行する必要がある。これをいくつかのケースで実現した

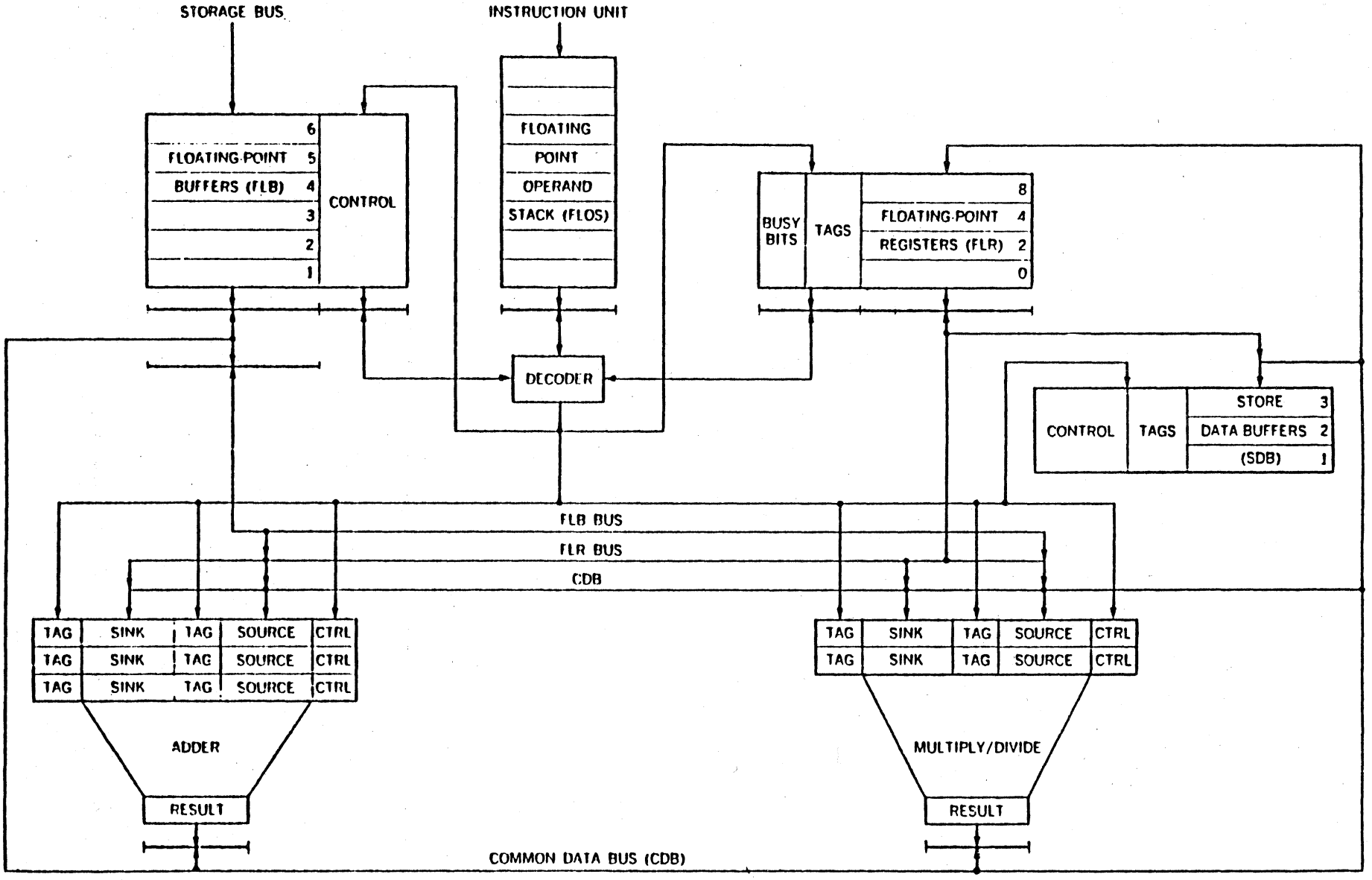


FIGURE 10. Floating point unit of IBM Model 91 with CDB and reservation stations.

のがＩＢＭ３６０／９１であるが，この制御方式はハードウェアをかけたほど効果が出ず，逐次処理アーキテクチャとしての並列化としては限界であったと思われる。逆にこの反省として並列処理アーキテクチャ（ベクトルアーキテクチャ）が必要となったともいえる。上記のような制御では，大量に命令を先取りするための命令バッファ，及びデータを先行して持ち込むためのオペランドバッファが必要である。これらのバッファを１まとめにして設置したのがＩＢＭ３６０／８５のバッファストーレッジ（キャッシュ）である。これは複数回使用するデータを保持することにより，メモリアクセスタイムが実効的に大幅に短縮され実効性能も大幅に改善された。最近の機械（富士通ＦＡＣＯＭ　Ｍ－７８０など）では命令用とデータ用の２組のキャッシュが置かれるようになりさらに高速化が図られている。これはまさに歴史が繰り返されていることになるが，回路素子の高集積化が進展したことにより実現可能となったものである。

1.3　競合

ハードウェア設計上での並列化に伴う最大の考慮点は，レジスタ，メモリ，バス，演算器などの資源使用率の向上である。逆に言えば，資源の使用要求がぶつかった時如何に競合

を整理調整して，本来実行するべき順序に並べてやるかである。バス，演算器などは使用するタイミングの調整だけであるが，レジスタ，メモリは内容の変更を伴うため結果がプログラム上で逐次処理したものと等しくなるようにしなければならず，順序関係が特に重要となる。

この制御には資源管理用のテーブルを持つことが多く，論理的な資源を物理的な中間資源に変換し管理する（ＴＡＧ）方式もある。

2. 並列（ベクトル）アーキテクチャ

上記のような逐次処理アーキテクチャのもとではハードウェア設計上の工夫での並列化には限度があり，うまくできても局所的に特殊ケースでの性能向上しか実現できなかった。科学技術計算の処理能力向上させるには，並列処理の程度を向上させることが効果的であり，論理的アーキテクチャ（ユーザからの機械の見え方）からの並列化を導入することが考えられた。これがいわゆるベクトルアーキテクチャであり，初期にはハードウェアの実現が困難であったものの，１９７０年代の半ばからはベクトルアーキテクチャを実現したスーパーコンピュータが登場した。

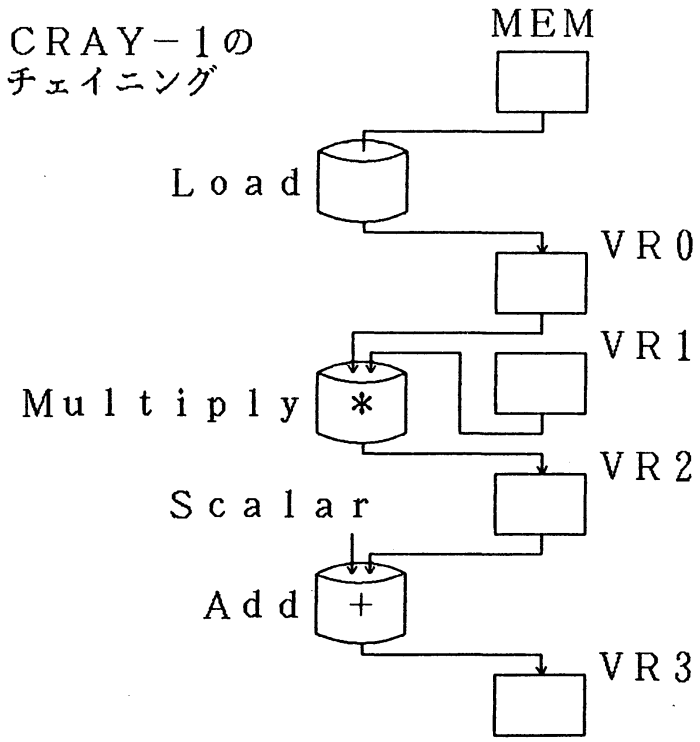
2.1 ベクトル命令

ベクトルアーキテクチャでは，ベクトル命令を持つことが逐次アーキテクチャとの大きな違いである。ベクトル命令は商用機ではパイプライン演算器で実行される。初期の機械ではベクトル命令はメモリーメモリ命令であり，メモリにおかれたベクトルオペランド同志をロードし，パイプライン演算器で演算し，その結果をまたメモリにストアするまでを１命令で実行した。この形式の命令では，メモリオペランドの競合によりデータのロード，ストア時に待ちが生じる可能性があり，デッドロック解消のための大量データバッファが必要である。またメモリオペランドの競合があるため，競合検出ハードウェアの量の問題で複数命令間での並列実行は殆ど不可能である。逐次アーキテクチャでも言えることであるが，一旦レジスタを仲介にしたメモリーレジスタ命令，レジスタ－レジスタ演算命令を設置するとレジスタオペランド競合条件の検出が比較的簡単となり，複数命令間での並列実行が可能となる。こういう理由から現状の商用機では，ベクトルレジスタを置くことが多い。

2.2　ベクトルレジスタ

ＦＡＣＯＭ　２３０－７５　ＡＰＵシステムではベクトルレジスタを設置した。この目的は小容量高速のＲＡＭを中間

データや定数として保持するために使用するものであり，いわば逐次型ＣＰＵのキャッシュに相当するものであった。これと同一目的で，上記プログラマブルなベクトルレジスタがＣＲＡＹ－１で実施された。このベクトルレジスタではベクトルの長さをベクトルレジスタの長さ以内の数に限定すること及び上述のメモリアクセス命令とレジスタ演算命令の分離とによって，複数命令間での並列実行が容易となった。またメモリオペランドの中間データとして，データバッファとしての役割も担っている。さらに複数命令での並列実行には，ＣＲＡＹ－１のチェイニング機構が有効であり，演算パイプライン間のデータの受け渡しをベクトルレジスタ経由で行っている。類似の機能が採用されている機械は多い。

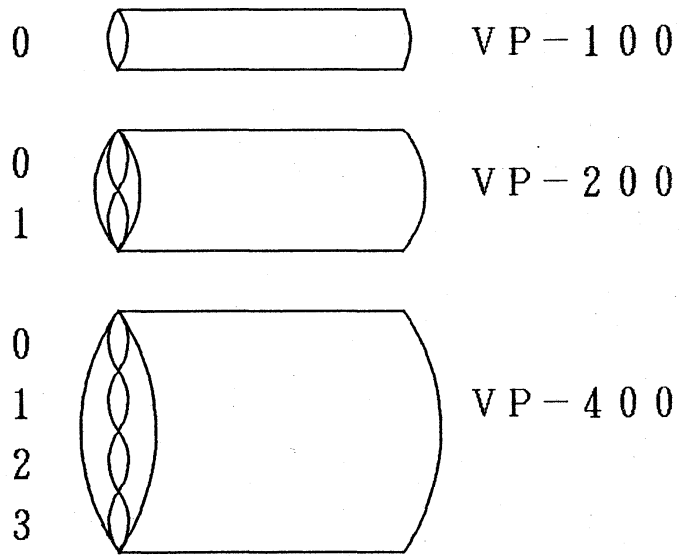


2.3　多重パイプライン

現状のスーパーコンピュータでは，複数のパイプライン演算器が設置されている。ＦＡＣＯＭ　ＶＰシリーズでは，異なる演算（乗算，加算，除算）用の複数のパイプラインのみならず，各演算パイプラインもさらに多重に置かれている。ＶＰシリーズでは，ハードウェアの自動制御によりこれらの多重パイプラインを同時に動作させており，物理的なパイプライン本数をプログラムで意識する必要はない。

物理的な多重パイプライン

ＶＰ－１００／２００／４００



2.4　条件付演算のベクトル化

ＦＡＣＯＭ　ＶＰシリーズでは，ＩＦ文のベクトル化を最

初から意識してアーキテクチャ及びコンパイラの設計を行った。ＩＦ文の自動ベクトライズを行うためのハードウェア機構は，(1)マスク機能(2)集数命令(3)間接アドレス命令によるメモリアクセス命令であり，その後の機械にはこれらの機能を採用したものが多い。コンパイラはＩＦ文のベクトル化をこれらのハードウェアとうまく利用して自動ベクトル化を行っている。

2.5　リカーランス

リカーランスとは，例えば

$$X_i = C_i + A_i * X_{i-1}$$

のような演算であり，１つ前のインデックスで示される変数の値を現在のインデックスの演算に使うような演算である。これはパイプライン演算器とは相性が悪いため，性能的には十分でない。このため演算アルゴリズムの変更が望まれる。ベクトル化したデータ同志を，ベクトル命令間の演算として上記リカーランス演算を実行できるように演算順序を変えるなどの工夫をすれば，演算器の使用率ひいては性能の向上を図ることができる。

3.　今後の展望

ベクトルプロセッサとしては抜本的なアーキテクチャ上の

改善は期待できないため，素子の改良以上の性能向上にはプロセッサ台数の増加が本命であろう。現状の単一台数のベクトルプロセッサのソフトウェアの充実（ライブラリ，アプリケーション）も今後の大きな課題ではあるが，さらに複数台プロセッサシステムの制御論理は大きな研究課題であろう。